Assignment 4

Concepts, intuitions and big picture

Multiple-choice questions

1. Select the sentence that best describes the terms “model”, “architecture”, and “weights”.

   a. Model and weights are the same; they form a succession of mathematical functions to build an architecture.

   b. An architecture is a succession of mathematical functions to build a model and its weights are those functions parameters.

   Answer: b

2. During forward propagation, in the forward function for a layer l you need to know what is the activation function in layer l (Sigmoid, tanh, ReLU, etc.). During Backpropagation, the corresponding backward function does not need to know the activation function for layer l since the gradient does not depends on it.

   a. True

   b. False

   Answer: b

3. You have built a network using the tanh activation for all the hidden units. You initialize the weights to relative large values, using randn(..,..)*1000. What will happen?

   a. It doesn’t matter. So long as you initialize the weights randomly gradient descent is not affected by whether the weights are large or small.

   b. This will cause the inputs of the tanh to also be very large, thus causing gradients to also become large. You therefore have to set the learning rate to be very small to prevent divergence; this will slow down learning.

   c. This will cause the inputs of the tanh to also be very large, causing the units to be “highly activated” and thus speed up learning compared to if the weights had to start from small values.

   d. This will cause the inputs of the tanh to also be very large, thus causing gradients to be close to zero. The optimization algorithm will thus become slow.

Answer: d

4. What is the “cache" used for in our implementation of forward propagation and backward propagation?

   a. It is used to cache the intermediate values of the cost function during training.

   b. We use it to pass variables computed during forward propagation to the corresponding backward propagation step. It contains useful values for backward propagation to compute derivatives.

   c. It is used to keep track of the hyperparameters that we are searching over, to speed up computation.

   d. We use it to pass variables computed during backward propagation to the corresponding forward propagation step. It contains useful values for forward propagation to compute activations.

   Answer: b

5. Among the following, which ones are "hyperparameters"? (Check all that apply.)

   a. size of the hidden layers.

   b. learning rate

   c. number of iterations

   d. number of layers in the neural network

   Answer: a, b, c, d

6. True/False? Vectorization allows you to compute forward propagation in an L-layer neural network without an explicit for-loop (or any other explicit iterative loop) over the layers $l = 1, 2, \ldots, L$.

   a. True

   b. False

   Answer: b

7. True or false? A language model usually does not need labels annotated by humans for its pretraining.

   a. True

   b. False

Answer: a

8. What is the order of the language modeling pipeline?

   a. First, the model, which handles text and returns raw predictions. The tokenizer then makes sense of these predictions and converts them back to text when needed.

   b. First, the tokenizer, which handles text and returns IDs. The model handles these IDs and outputs a prediction, which can be some text.

   c. The tokenizer handles text and returns IDs. The model handles these IDs and outputs a prediction. The tokenizer can then be used once again to convert these predictions back to some text.

   Answer: c

Short answer questions

1. Look at the definition of [Stochastic] Gradient Descent in PyTorch: https://pytorch.org/docs/stable/generated/torch.optim.SGD.html. You will see that this is a bit more complex than what we have seen in the class. Let's understand a few nuances here.

   a. Notice the maximize parameter which controls whether we are running a maximization or minimization. In the algorithm the effect of this parameter is shown as essentially a change in the sign of the gradients:

   if $maximize$

   $$\theta_t \leftarrow \theta_{t-1} + \gamma g_t$$

   else

   $$\theta_t \leftarrow \theta_{t-1} - \gamma g_t$$

   How do you think this change leads to the desired outcome (maximization vs minimization)?

   Answer: Based on the algorithm above, when `maximize` is `True`, the update rule adds the gradient to θ_{t-1}, moving the parameters up the slope, instead of down. In other words, it performs gradient ascent instead of gradient descent. When `maximize` ia `False`, the gradient is subtracted from θ_{t-1}, resulting in gradient descent (or minimization).

   b. The next set of parameters `momentum, dampening, weight_decay`. What do you think the impact of these parameters are? Read the documentations and interpret their roles.

   Answer:

- **Momentum:** This is used to accelerate the gradient descent in the relevant direction (i.e. maximization or minimization) by setting a fraction of the update vector from the previous step that should be added to the current update vector. This allows the optimizer move faster in the relevant direction with consistent gradients that smooth out changes in gradient noise or direction.

- **Dampening:** This parameter is used in conjunction with momentum to cancel out (or dampen) some of the effects of the momentum term. It reduces the contribution of the previous momentum vector to the current update.

- **Weight Decay:** This parameter is used to implement L2 regularization, which is commonly used to prevent overfitting during the training of neural networks. This works by adding a penalty term to the loss function that further reduces the values of the model's weights, especially large weights. This makes the model's weights smaller, making them less sensitive to specific input values in the training data, thus reducing overfitting to training data and making the model more likely to generalize to unseen data.

2. What are few benefits to using Fixed-Window-LM over n-grams language models? (limit your answer to less then 5 sentences).

   Answer: Fixed-Window LMs are based on neural network architectures, which enable them to capture more complex relationships between words in their fixed windows. Unlike n-gram models that are susceptible to the sparsity problem and assign zero probabilities to sequences that were not encountered during training, Fixed-Window LMs learn distributed representations (or embeddings) of words and are capable of handling new variations of word sequences more effectively based on the similarity of the words. As such, Fixed-Window LMs are able to generalize better to new text.

3. When searching over hyperparameters, a typical recommendation is to do random search rather than a systematic grid search. Why do you think that is the case? (less than 2 sentences).

   Answer: This is because by randomly sampling values for the hyperparameters you intend to tune, random search covers a wider range of values relative to the number of trials, compared to a grid search approach that goes based on a predefined set of values (or grid). As such, random search is more likely to find more optimal values, unlike grid search, which will definitely miss any optimal values that are not in the predefined value space.

4. Remember the normalization layer that we saw during the class. Here is the corresponding PyTorch page: https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html. In the normalization formula, why do we use epsilon ϵ in the denominator?

Answer: The addition of epsilon (ϵ) in the denominator is to prevent division by zero or very small numbers in cases where the variance is too close to zero. Such divisions can lead to overflow, where the resulting number is undefined (NaN) or too large to be represented by the floating-point type (Inf). Simply put, adding epsilon (ϵ) ensures numerical stability in the normalization formula.

Softmax Smackdown: Squishing the Competition

Softmax gradient

You might remember in the midterm exam that we saw a neural network with a Softmax and a cross-entropy loss:

$\hat{\mathbf{y}} = \text{Softmax}(\mathbf{h})$, $J = \text{CE}(\mathbf{y}, \hat{\mathbf{y}})$,
$\mathbf{h}, \mathbf{y}, \hat{\mathbf{y}} \in \mathbb{R}^d$.

Basically here $\hat{\mathbf{y}}$ is a d-dimensional probability distribution over d possible options (i.e. $\sum_i \mathbf{y}_i = 1$ and $\forall i : \mathbf{y}_i \in [0, 1]$). $\mathbf{y}$ is a d-dimensional one-hot vector, i.e., all of these values are zeros except one that corresponds to the correct label.

Here we want to prove the hint that was provided in the exam, which was:

$\nabla_{\mathbf{h}} J = \hat{\mathbf{y}} - \mathbf{y}$.

Prove the above statement. In your proof, use the statement that you proved in HW3 about the gradient of Softmax function: $\frac{\partial \hat{\mathbf{y}}_j}{\partial h_i} = \hat{\mathbf{y}}_i(\delta_{ij} - \hat{\mathbf{y}}_j)$, where δ_{ij} is the Kronecker delta function and $\hat{\mathbf{y}}_i$ is the value in the i-th index of $\hat{\mathbf{y}}$.

Answer:

For loss function as the cross-entropy:

$J = -\sum_k y_k \log(\hat{\mathbf{y}}_k)$

Compute $\frac{\partial J}{\partial h_i}$ for each element h_i of the input vector $\mathbf{h}$.

Using the chain rule:

$\frac{\partial J}{\partial h_i} = \sum_j \frac{\partial J}{\partial \hat{\mathbf{y}}_j} \frac{\partial \hat{\mathbf{y}}_j}{\partial h_i}$

From the cross-entropy loss, the partial derivative with respect to $\hat{\mathbf{y}}_j$ is:

$\frac{\partial J}{\partial \hat{\mathbf{y}}_j} = -\frac{y_j}{\hat{\mathbf{y}}_j}$

Substituting this and the given Softmax gradient $\frac{\partial \hat{\mathbf{y}}_j}{\partial h_i} = \hat{\mathbf{y}}_i(\delta_{ij} - \hat{\mathbf{y}}_j)$ into the chain rule:

$\frac{\partial J}{\partial h_i} = \sum_j \left(-\frac{y_j}{\hat{\mathbf{y}}_j}\right) \left(\hat{\mathbf{y}}_i(\delta_{ij} - \hat{\mathbf{y}}_j)\right)$

$\frac{\partial J}{\partial h_i} = -\hat{\mathbf{y}}_i \sum_j \frac{y_j}{\hat{\mathbf{y}}_j}(\delta_{ij} - \hat{\mathbf{y}}_j)$

$\frac{\partial J}{\partial h_i} = -\hat{\mathbf{y}}_i \sum_j \left(\frac{y_j \delta_{ij}}{\hat{\mathbf{y}}_j} - y_j\right)$

To evaluate the sum...

Due to the Kronecker delta δ_{ij}, the term $\frac{y_j \delta_{ij}}{\hat{\mathbf{y}}_j}$ is non-zero only when $j = i$. So:

$\sum_j \frac{y_j \delta_{ij}}{\hat{\mathbf{y}}_j} = \frac{y_i \delta_{ii}}{\hat{\mathbf{y}}_i} = \frac{y_i}{\hat{\mathbf{y}}_i}$.

The sum $\sum_j y_j$ is the sum of elements in the one-hot vector $\mathbf{y}$. Since $\mathbf{y}$ is a one-hot vector:

$\sum_j y_j = 1$.

Substituting these simplified sums back to the earlier equation:

$\frac{\partial J}{\partial h_i} = -\hat{\mathbf{y}}_i \left(\frac{y_i}{\hat{\mathbf{y}}_i} - 1\right)$

$\frac{\partial J}{\partial h_i} = -y_i + \hat{\mathbf{y}}_i$

$\frac{\partial J}{\partial h_i} = \hat{\mathbf{y}}_i - y_i$

Since this holds for each element i, the gradient vector:

$\nabla_{\mathbf{h}} J = \hat{\mathbf{y}} - \mathbf{y}$

Softmax temperature

Remember the softmax function, $\sigma(\mathbf{z})$? Here we will add a temperature parameter $\tau \in \mathbb{R}^+$ to this function:

Softmax: $\sigma(\mathbf{z}; \tau)_i = \frac{e^{z_i/\tau}}{\sum_{j=1}^{K} e^{z_j/\tau}}$ for $i = 1, \ldots, K$

Show the following:

1. In the limit as temperature goes to zero $\tau \to 0$, softmax becomes the same as greedy action selection, argmax.

   Answer:

   The Softmax function with temperature is:

   $\sigma(\mathbf{z}; \tau)_i = \frac{e^{z_i/\tau}}{\sum_{j=1}^{K} e^{z_j/\tau}}$

   Let $m = \max(z_j)$ be the maximum value in the input vector $\mathbf{z}$. By factoring out $e^{m/\tau}$:

$$\sigma(\mathbf{z};\tau)_i = \frac{e^{z_i/\tau}}{e^{m/\tau}\sum_{j=1}^{K} e^{(z_j-m)/\tau}} = \frac{e^{(z_i-m)/\tau}}{\sum_{j=1}^{K} e^{(z_j-m)/\tau}}$$

For the denominator, consider the limit as $\tau \to 0^+$ for the term $e^{(z_j-m)/\tau}$:

- If $z_j < m$, then $(z_j - m) < 0$. As $\tau \to 0^+$, $(z_j - m)/\tau \to -\infty$. Thus, $e^{(z_j-m)/\tau} \to 0$.
- If $z_j = m$, then $(z_j - m) = 0$. As $\tau \to 0^+$, $(z_j - m)/\tau = 0$. Thus, $e^{(z_j-m)/\tau} = e^0 = 1$.

Let $I = \{i \mid z_i = m\}$ be the set of indices where z_i is maximum. The sum in the denominator approaches the number of maximum values:

$\sum_{j=1}^{K} e^{(z_j-m)/\tau} \to \sum_{j\in I} 1 + \sum_{j\notin I} 0 = |I|$

For the numerator, as $\tau \to 0^+$:

- If $z_i = m$ (i.e., $i \in I$), $e^{(z_i-m)/\tau} \to 1$.
- If $z_i < m$ (i.e., $i \notin I$), $e^{(z_i-m)/\tau} \to 0$.

Combining these, as $\tau \to 0^+$:

$$\sigma(\mathbf{z};\tau)_i \to \begin{cases} \frac{1}{|I|} & \text{if } z_i = m \\ 0 & \text{if } z_i < m \end{cases}$$

If there's only one occurence of the maximum ($|I| = 1$), this becomes 1 for the index of the maximum and 0 for others, which is essentially the argmax function.

2. In the limit as temperature goes to infinity $\tau \to +\infty$, softmax gives equiprobable selection among all actions.

   Answer:

   Consider the term z_i/τ. As $\tau \to +\infty$, $z_i/\tau \to 0$ for any finite z_i.

   So, $e^{z_i/\tau} \to e^0 = 1$ for all $i = 1, \ldots, K$.

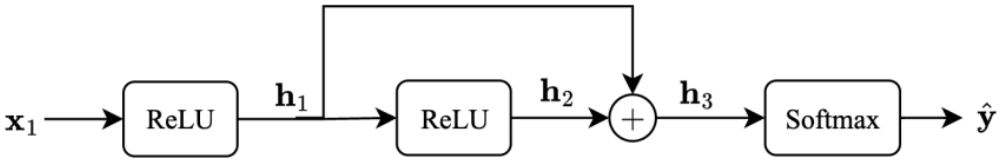
   The Softmax formula becomes:

   $$\sigma(\mathbf{z};\tau)_i = \frac{e^{z_i/\tau}}{\sum_{j=1}^{K} e^{z_j/\tau}} \to \frac{1}{\sum_{j=1}^{K} 1} = \frac{1}{K}$$

   As $\tau \to +\infty$, the probability for each action approaches $1/K$. This means all actions become equally probable, i.e. "equiprobable selection among all actions".

Backprop Through Residual Connections

As you know, When neural networks become very deep (i.e. have many layers), they become difficult to train due to the vanishing gradient problem - as the gradient is back-propagated through many layers, repeated multiplication can make the gradient extremely small, so that performance plateaus or even degrades. An effective approach is to add skip connections that skip one or more layers. See the provided network.



$$\mathbf{x} \in \mathbb{R}^d,\ \mathbf{W}_{1,2} \in \mathbb{R}^{d \times d},\ \hat{\mathbf{y}} \in \mathbb{R}^d$$

$$\mathbf{z}_1 = \mathbf{W}_1 \mathbf{x}_1,\ \mathbf{h}_1 = \mathrm{ReLU}(\mathbf{z}_1),$$

$$\mathbf{z}_2 = \mathbf{W}_2 \mathbf{h}_1,\ \mathbf{h}_2 = \mathrm{ReLU}(\mathbf{z}_2),$$

$$\mathbf{h}_3 = \mathbf{h}_1 + \mathbf{h}_2,\ \hat{\mathbf{y}} = \mathrm{Softmax}(\mathbf{h}_3),\ J = \mathrm{CE}(\mathbf{y}, \hat{\mathbf{y}})$$

1. You have seen this in the quiz, but lets try again. Prove that:
   $\frac{\partial}{\partial z_i}\mathrm{ReLU}(z) = \mathbf{1}\{z_i > 0\}$

   where $\mathbf{1}\{x > 0\} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$. More generally,
   $\nabla_{\mathbf{z}}\mathrm{ReLU}(\mathbf{z}) = \mathrm{diag}(\mathbf{1}\{\mathbf{z} > 0\})$ where $\mathbf{1}\{\cdot\}$ is applied per each dimension and $\mathrm{diag}(\cdot)$ turns a vector into a diagonal matrix.

   Answer:

   $\mathrm{ReLU}(z) = \max(0, z).$

   For a single input z_i:

   $\mathrm{ReLU}(z_i) = \max(0, z_i).$

   For the derivative of $\mathrm{ReLU}(z_i)$ with respect to z_i:

   - Case 1: If $z_i > 0$, $\mathrm{ReLU}(z_i) = z_i$.

     The derivative with respect to z_i is:

     $\frac{\partial}{\partial z_i}\mathrm{ReLU}(z_i) = \frac{\partial}{\partial z_i}(z_i) = 1.$

   - Case 2: If $z_i \leq 0$, $\mathrm{ReLU}(z_i) = 0$.

     The derivative with respect to z_i for $z_i < 0$ is:

     $\frac{\partial}{\partial z_i}\mathrm{ReLU}(z_i) = \frac{\partial}{\partial z_i}(0) = 0.$

   Note: The ReLU function is technically not differentiable at $z_i = 0$. The derivative is only typically defined as 0 at this point for practical reasons.

   Combining these cases, the derivative of $\mathrm{ReLU}(z_i)$ with respect to z_i is:

   $$\frac{\partial}{\partial z_i}\mathrm{ReLU}(z_i) = \begin{cases} 1 & \text{if } z_i > 0 \\ 0 & \text{if } z_i \leq 0 \end{cases}$$

   This is exactly the definition of the indicator function $\mathbf{1}\{z_i > 0\}$.

Therefore, $\frac{\partial}{\partial z_i}\mathrm{ReLU}(z_i) = \mathbf{1}\{z_i > 0\}$.

For the vectorized case, $\mathrm{ReLU}(\mathbf{z})$ applies the ReLU function element-wise to the vector $\mathbf{z}$. The gradient $\nabla_{\mathbf{z}}\mathrm{ReLU}(\mathbf{z})$ is a vector where each element is the partial derivative with respect to the corresponding element of $\mathbf{z}$.

$$\nabla_{\mathbf{z}}\mathrm{ReLU}(\mathbf{z}) = \begin{pmatrix} \frac{\partial}{\partial z_1}\mathrm{ReLU}(z_1) \\ \frac{\partial}{\partial z_2}\mathrm{ReLU}(z_2) \\ \vdots \\ \frac{\partial}{\partial z_d}\mathrm{ReLU}(z_d) \end{pmatrix} = \begin{pmatrix} \mathbf{1}\{z_1 > 0\} \\ \mathbf{1}\{z_2 > 0\} \\ \vdots \\ \mathbf{1}\{z_d > 0\} \end{pmatrix}$$

This vector can be represented as a diagonal matrix where the diagonal elements are the elements of the vector $\mathbf{1}\{\mathbf{z} > 0\}$, and all off-diagonal elements are zero. This is the definition of $\mathrm{diag}(\mathbf{1}\{\mathbf{z} > 0\})$. Thus:

$$\nabla_{\mathbf{z}}\mathrm{ReLU}(\mathbf{z}) = \mathrm{diag}(\mathbf{1}\{\mathbf{z} > 0\}).$$

2. As you see, a single variable ($\mathbf{h}_2$ in the example) feeds into two different layers of the network. Here we want to show that the two upstream signals stemming from this node are merged as summation during Backprop. Specifically prove that:

$$\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_1} = \mathbf{I} + \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2}\frac{\partial \mathbf{z}_2}{\partial \mathbf{h}_1}$$

Answer:

Assume $\mathbf{h}_3$ is computed by adding $\mathbf{h}_1$ to the output of a block of layers, where the output of the block is some function of $\mathbf{h}_1$, let's call it $f(\mathbf{h}_1)$. Here, the residual path from $\mathbf{h}_1$ goes through operations that result in $\mathbf{h}_2$ via $\mathbf{z}_2$, and the derivative of this path's output with respect to $\mathbf{h}_1$ is $\frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2}\frac{\partial \mathbf{z}_2}{\partial \mathbf{h}_1}$.

So, $\mathbf{h}_3 = \mathbf{h}_1 + f(\mathbf{h}_1)$.

Taking the derivative with respect to $\mathbf{h}_1$:

$$\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_1} = \frac{\partial}{\partial \mathbf{h}_1}(\mathbf{h}_1 + f(\mathbf{h}_1))$$

Using the sum rule:

$$\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_1} = \frac{\partial \mathbf{h}_1}{\partial \mathbf{h}_1} + \frac{\partial f(\mathbf{h}_1)}{\partial \mathbf{h}_1}$$

The derivative of a vector with respect to itself is the identity matrix $\mathbf{I}$:

$$\frac{\partial \mathbf{h}_1}{\partial \mathbf{h}_1} = \mathbf{I}$$

Substituting the given derivative for the residual path:

$$\frac{\partial f(\mathbf{h}_1)}{\partial \mathbf{h}_1} = \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2}\frac{\partial \mathbf{z}_2}{\partial \mathbf{h}_1}$$

Therefore, substituting back into the equation for $\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_1}$:

$\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_1} = \mathbf{I} + \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2} \frac{\partial \mathbf{z}_2}{\partial \mathbf{h}_1}$

This shows that the gradient $\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_1}$ is the sum of the gradient from the direct connection ($\mathbf{I}$) and the gradient from the path through the layers ($\frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2} \frac{\partial \mathbf{z}_2}{\partial \mathbf{h}_1}$). This summation of gradients is key to how residual connections help combat vanishing gradients during backpropagation.

3. In the given neural network your task is to compute the gradient $\frac{\partial J}{\partial \mathbf{x}}$. You are allowed (and highly encouraged) to use variables to represent intermediate gradients.

   Hint 1: Compute these gradients in order to build up your answer: $\frac{\partial J}{\partial \mathbf{h}_3}$, $\frac{\partial J}{\partial \mathbf{h}_2}$, $\frac{\partial J}{\partial \mathbf{z}_2}$, $\frac{\partial J}{\partial \mathbf{h}_1}$, $\frac{\partial J}{\partial \mathbf{z}_1}$, $\frac{\partial J}{\partial \mathbf{x}}$. Show your work so we are able to give partial credit!

   Hint 2: Recall that downstream = upstream * local.

   Answer:

We apply the chain rule backward from the loss J to $\mathbf{x}$:

1. $\frac{\partial J}{\partial \mathbf{h}_3} = \hat{\mathbf{y}} - \mathbf{y}$ (Gradient of CE with Softmax)
2. $\frac{\partial J}{\partial \mathbf{h}_2}$: From $\mathbf{h}_3 = \mathbf{h}_1 + \mathbf{h}_2$, $\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} = \mathbf{I}$.

   $\frac{\partial J}{\partial \mathbf{h}_2} = \frac{\partial J}{\partial \mathbf{h}_3} \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} = (\hat{\mathbf{y}} - \mathbf{y})\mathbf{I} = \hat{\mathbf{y}} - \mathbf{y}$
3. $\frac{\partial J}{\partial \mathbf{z}_2}$: From $\mathbf{h}_2 = \mathrm{ReLU}(\mathbf{z}_2)$, $\frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2} = \mathrm{diag}(\mathbf{1}\{\mathbf{z}_2 > 0\})$.

   $\frac{\partial J}{\partial \mathbf{z}_2} = \frac{\partial J}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2} = (\hat{\mathbf{y}} - \mathbf{y}) \cdot \mathrm{diag}(\mathbf{1}\{\mathbf{z}_2 > 0\})$
4. $\frac{\partial J}{\partial \mathbf{h}_1}$: $\mathbf{h}_1$ contributes to $\mathbf{h}_3$ directly and via the path through $\mathbf{z}_2$ and $\mathbf{h}_2$.

   Path 1 (direct to $\mathbf{h}_3$): $\frac{\partial J}{\partial \mathbf{h}_3} \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_1} = (\hat{\mathbf{y}} - \mathbf{y})\mathbf{I} = \hat{\mathbf{y}} - \mathbf{y}$

   Path 2 (via $\mathbf{z}_2, \mathbf{h}_2$): $\frac{\partial J}{\partial \mathbf{z}_2} \frac{\partial \mathbf{z}_2}{\partial \mathbf{h}_1}$. From $\mathbf{z}_2 = \mathbf{W}_2 \mathbf{h}_1$, $\frac{\partial \mathbf{z}_2}{\partial \mathbf{h}_1} = \mathbf{W}_2^T$.

   Gradient from Path 2: $\left((\hat{\mathbf{y}} - \mathbf{y}) \cdot \mathrm{diag}(\mathbf{1}\{\mathbf{z}_2 > 0\})\right) \mathbf{W}_2^T$

   Summing paths:
   $\frac{\partial J}{\partial \mathbf{h}_1} = (\hat{\mathbf{y}} - \mathbf{y}) + (\hat{\mathbf{y}} - \mathbf{y}) \cdot \mathrm{diag}(\mathbf{1}\{\mathbf{z}_2 > 0\}) \mathbf{W}_2^T = (\hat{\mathbf{y}} - \mathbf{y}) \left(\mathbf{I} + \mathrm{diag}(\mathbf{1}\{\mathbf{z}_2 > 0\} \right.$
5. $\frac{\partial J}{\partial \mathbf{z}_1}$: From $\mathbf{h}_1 = \mathrm{ReLU}(\mathbf{z}_1)$, $\frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1} = \mathrm{diag}(\mathbf{1}\{\mathbf{z}_1 > 0\})$.

   $\frac{\partial J}{\partial \mathbf{z}_1} = \frac{\partial J}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1} = \left((\hat{\mathbf{y}} - \mathbf{y}) \left(\mathbf{I} + \mathrm{diag}(\mathbf{1}\{\mathbf{z}_2 > 0\}) \mathbf{W}_2^T \right) \right) \cdot \mathrm{diag}(\mathbf{1}\{\mathbf{z}_1 > 0\})$

6. $\frac{\partial J}{\partial \mathbf{x}}$: From $\mathbf{z}_1 = \mathbf{W}_1\mathbf{x}$, $\frac{\partial \mathbf{z}_1}{\partial \mathbf{x}} = \mathbf{W}_1^T$.

   $\frac{\partial J}{\partial \mathbf{x}} = \frac{\partial J}{\partial \mathbf{z}_1}\frac{\partial \mathbf{z}_1}{\partial \mathbf{x}} = \left(\left((\hat{\mathbf{y}} - \mathbf{y})\left(\mathbf{I} + \text{diag}(\mathbf{1}\{\mathbf{z}_2 > 0\})\mathbf{W}_2^T\right)\right) \cdot \text{diag}(\mathbf{1}\{\mathbf{z}_1 > 0\})\right)\mathbf{W}_1^T$

4. Based on your derivations, explain why residual connections help mitigate vanishing gradients.

   Answer:

   Based on the derivation above, the gradient of the loss with respect to an earlier layer's output (like $\mathbf{h}_1$) includes an additive term corresponding to the residual connection:

   $\frac{\partial J}{\partial \mathbf{h}_1} = (\hat{\mathbf{y}} - \mathbf{y})\left(\mathbf{I} + \text{diag}(\mathbf{1}\{\mathbf{z}_2 > 0\})\mathbf{W}_2^T\right)$.

   The presence of the identity matrix ($\mathbf{I}$) within the gradient calculation is key in propagating the gradient forward. In traditional deep networks without residual connections, gradients are multiplied through many layers, which can cause them to shrink or vanish exponentially if the layer transformations have small singular values.

   With a residual connection, the gradient has a direct path back through the identity term. This means that even if the gradients flowing through the weighted layers of the residual block become small, the gradient can still propagate effectively through the additive identity path. This additive flow prevents the gradient from vanishing as quickly, thereby allowing for more stable and effective training of deeper neural networks. The residual connection essentially provides an alternative, unimpeded route for the gradient signal.

Programming